

Fig. 1 (prior art)

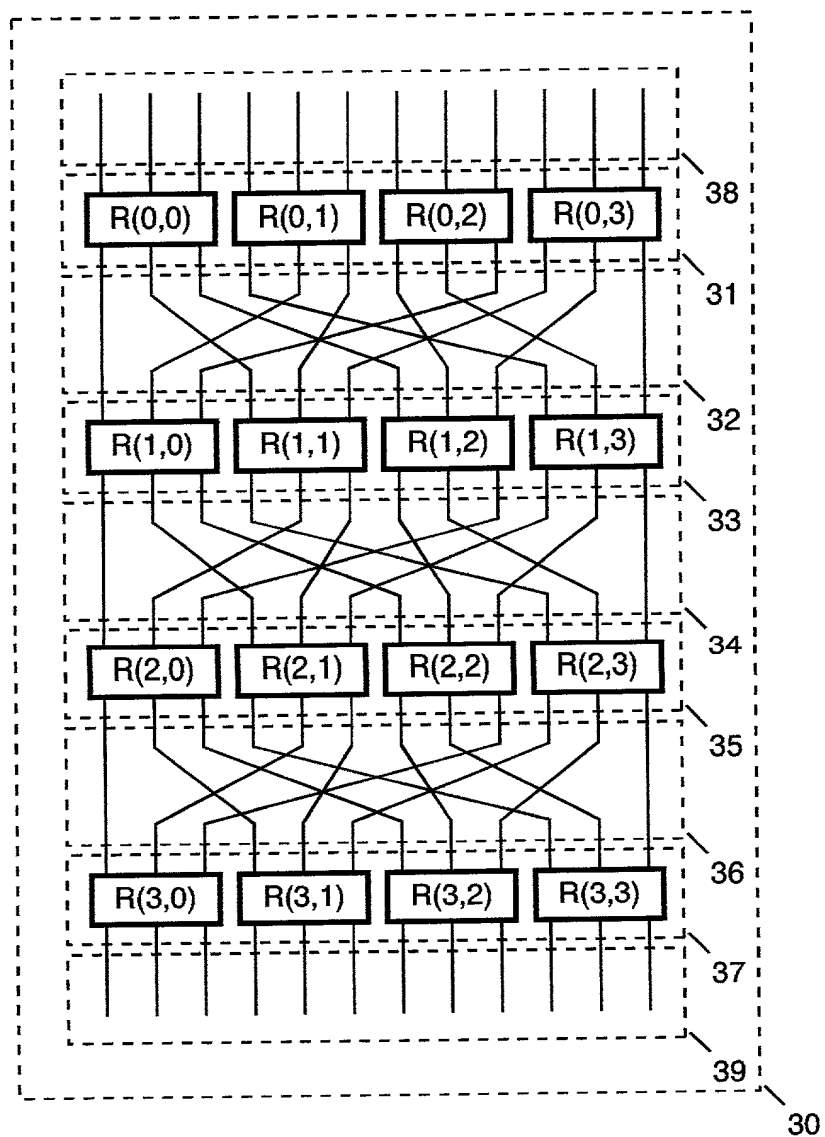


Fig. 2

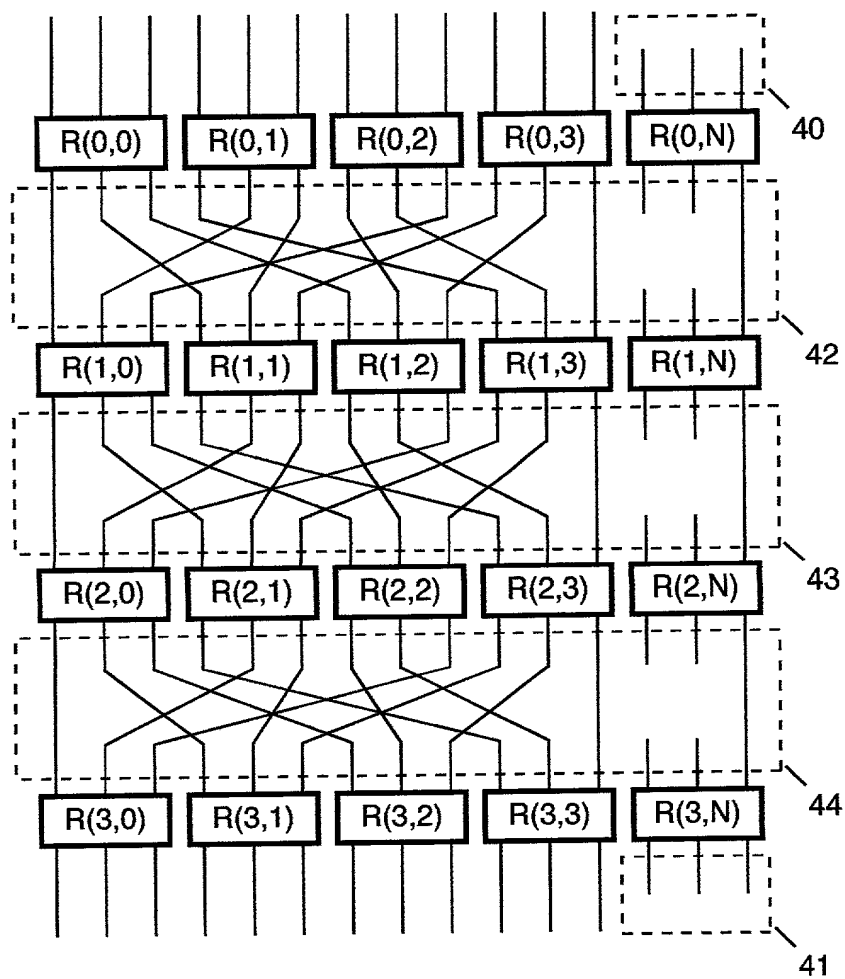


Fig. 3A

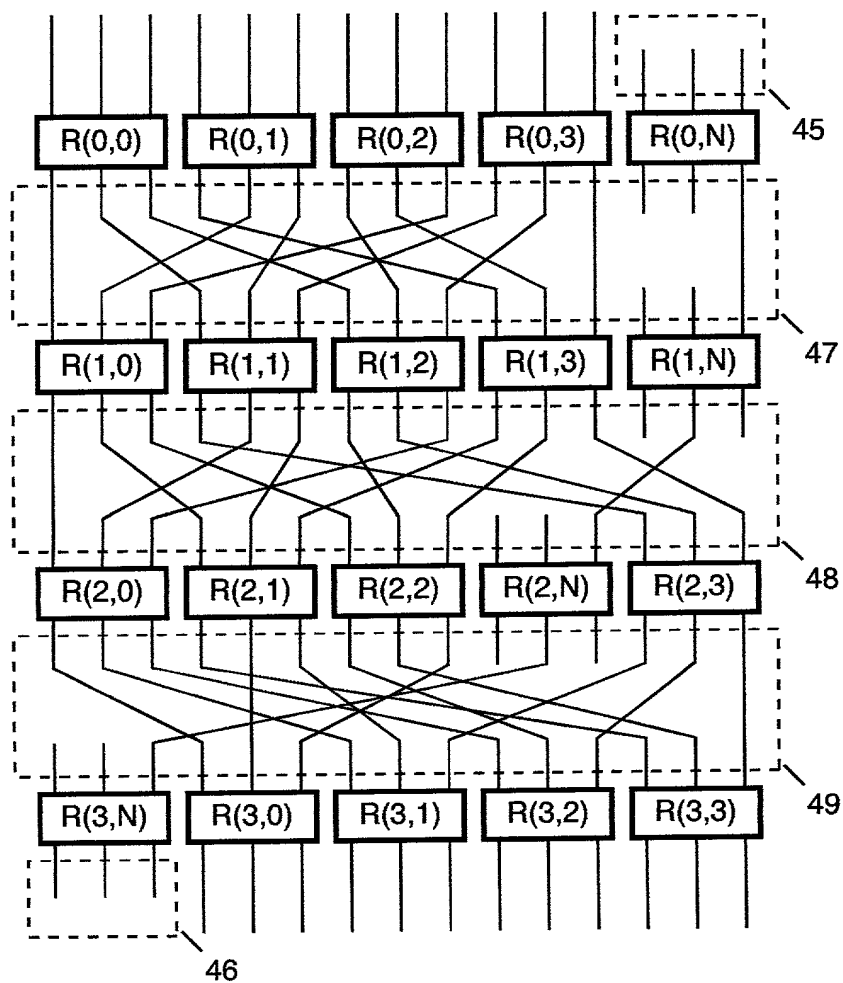


Fig. 3B

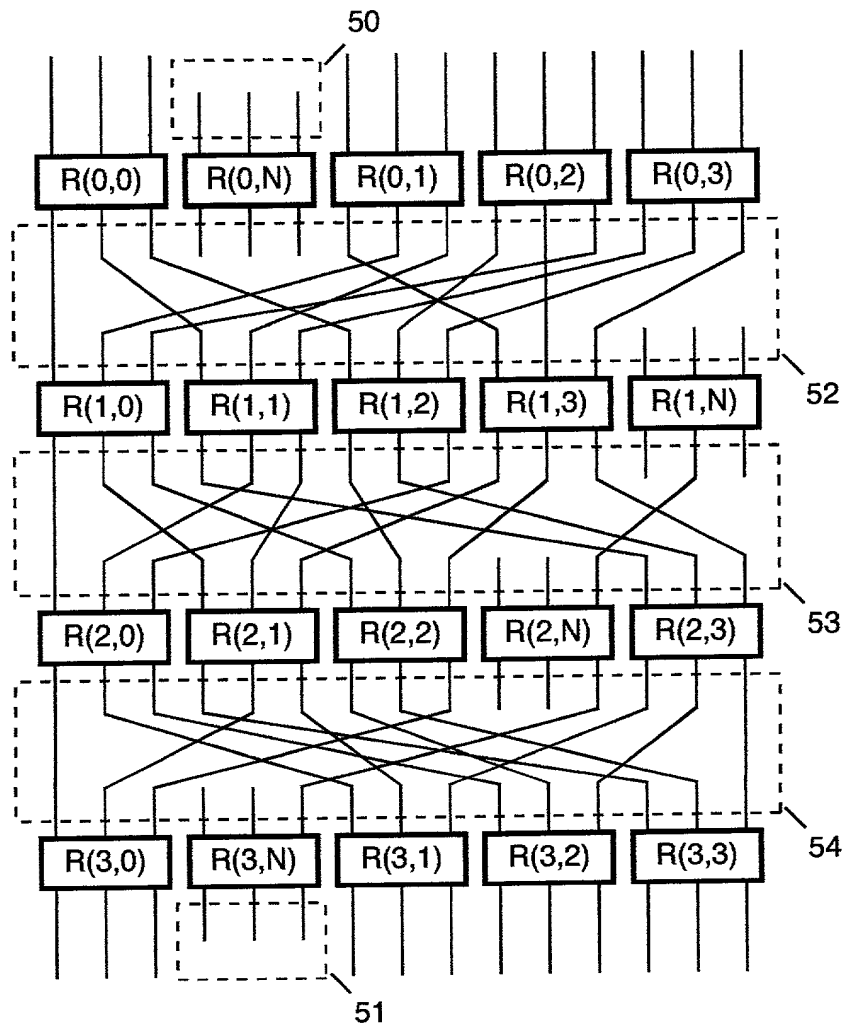


Fig. 3C

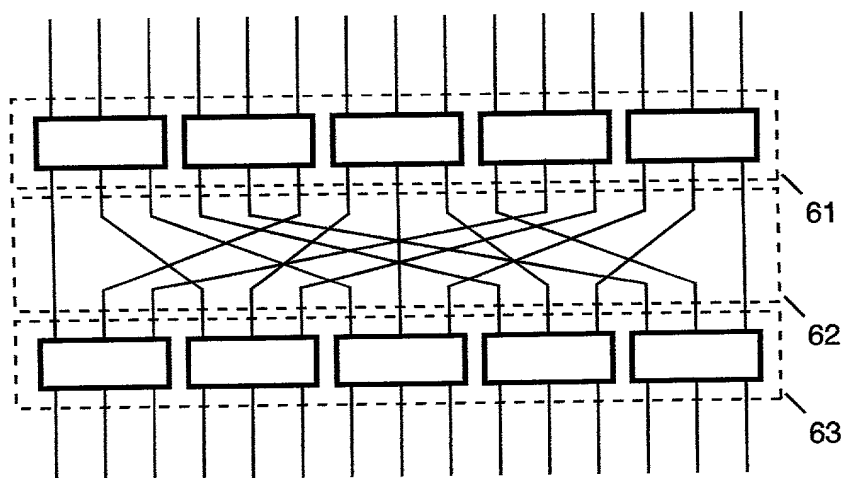


Fig. 4

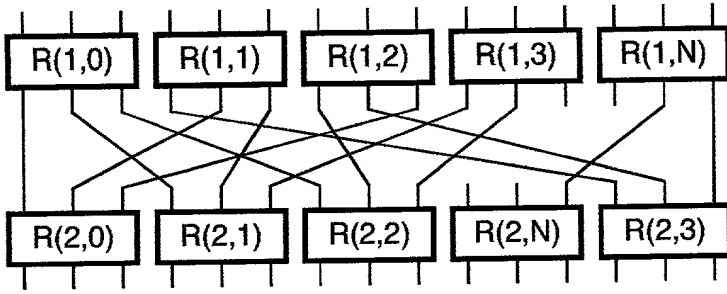


Fig. 5A

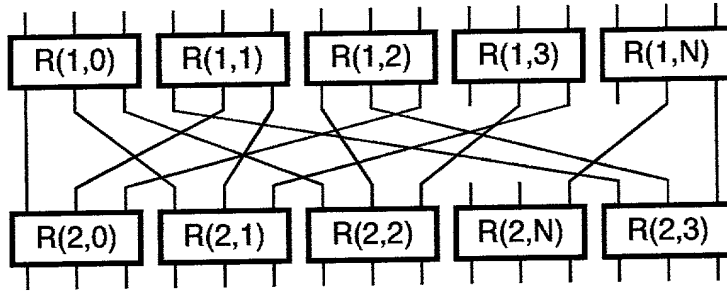


Fig. 5B

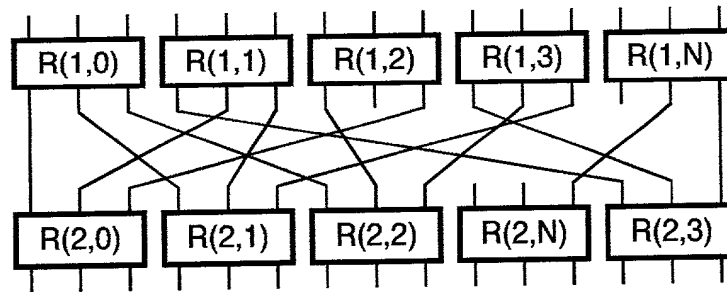


Fig. 5C

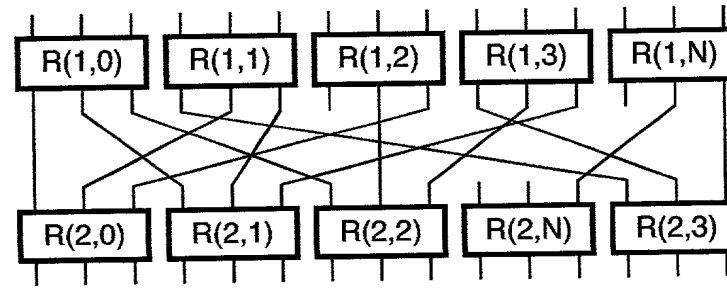


Fig. 5D

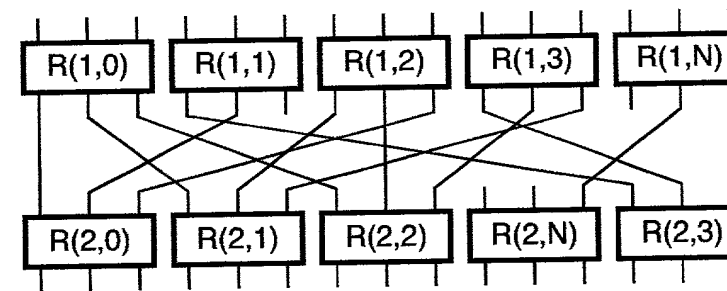


Fig. 5E

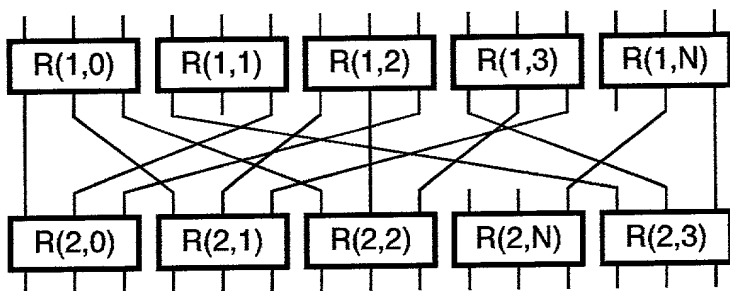


Fig. 5F

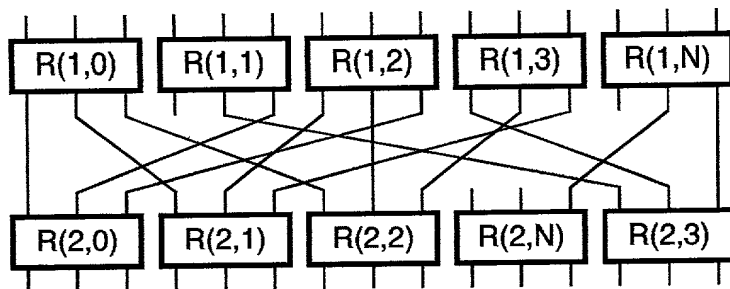


Fig. 5G

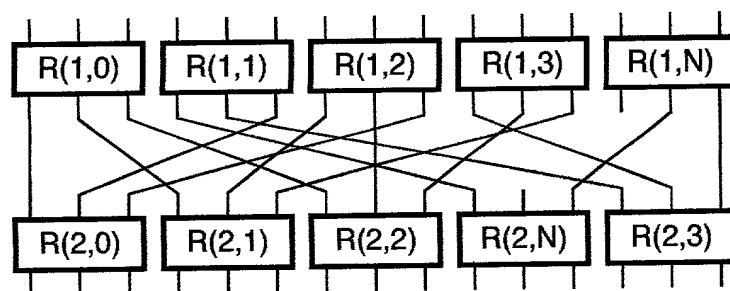


Fig. 5H

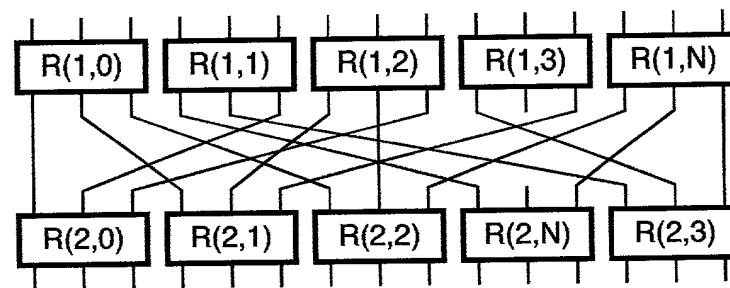


Fig. 5I

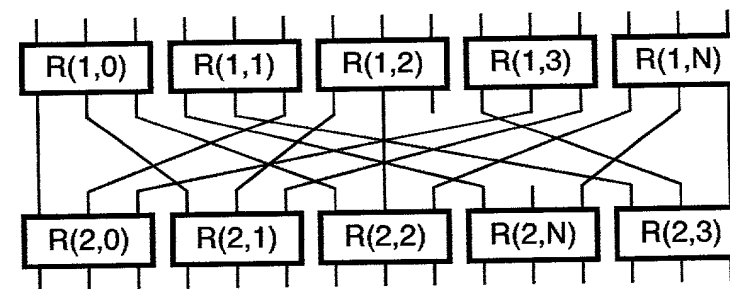


Fig. 5J

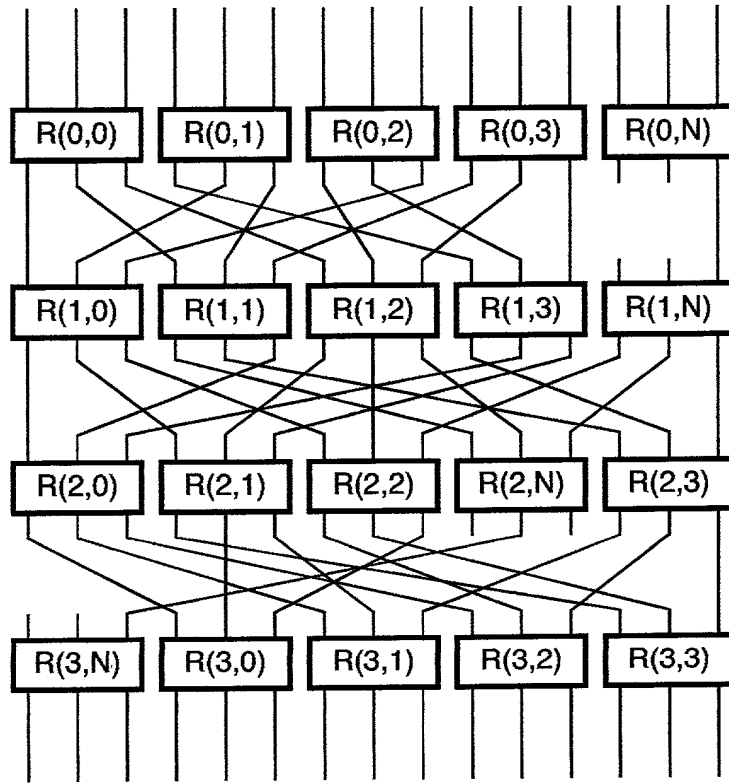


Fig. 5K

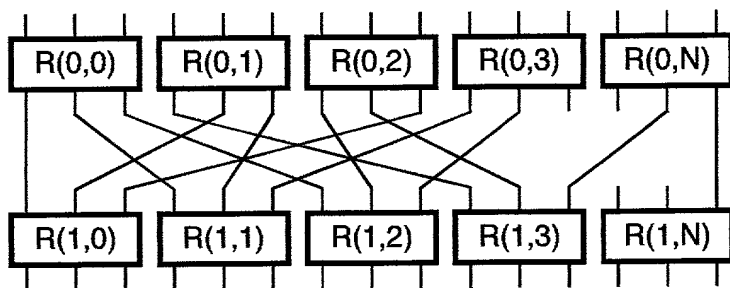


Fig. 6A

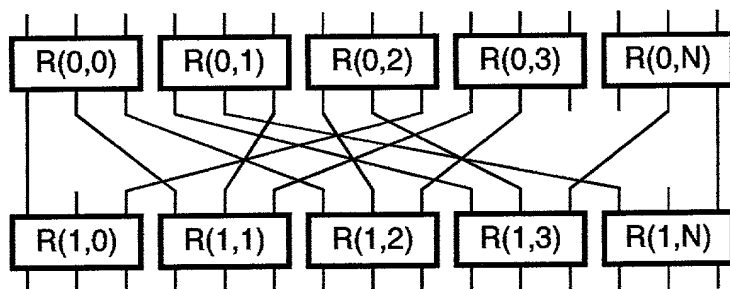


Fig. 6B

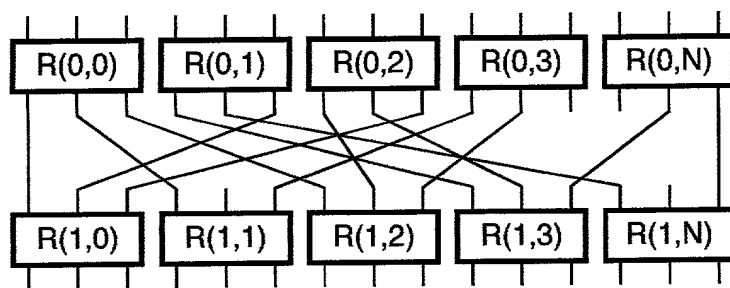


Fig. 6C

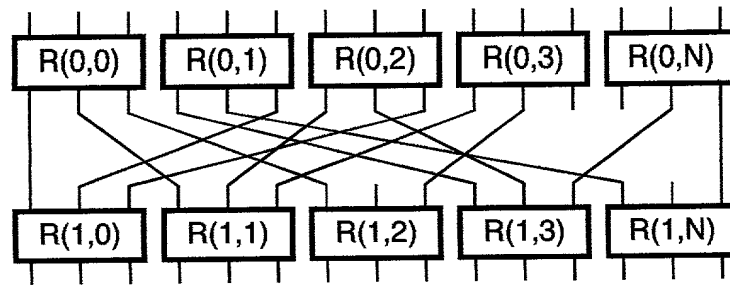


Fig. 6D

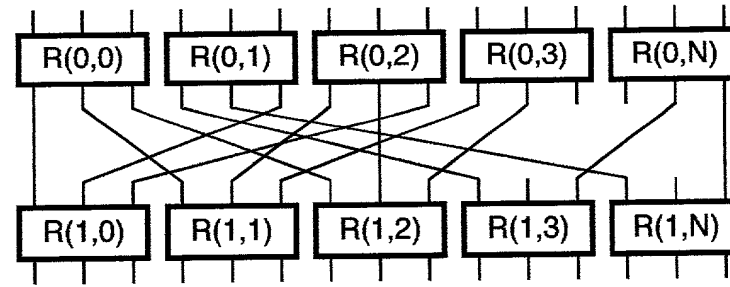


Fig. 6E

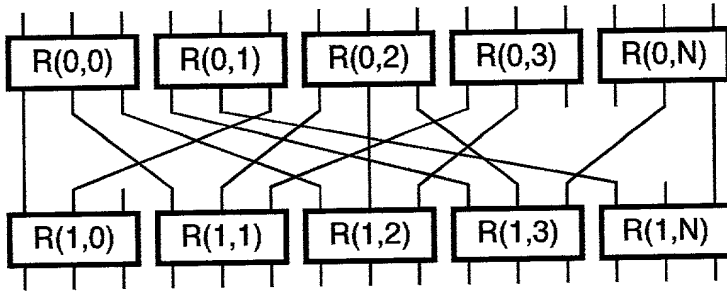


Fig. 6F

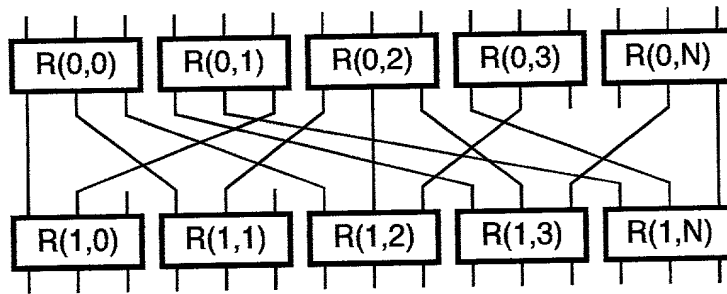


Fig. 6G

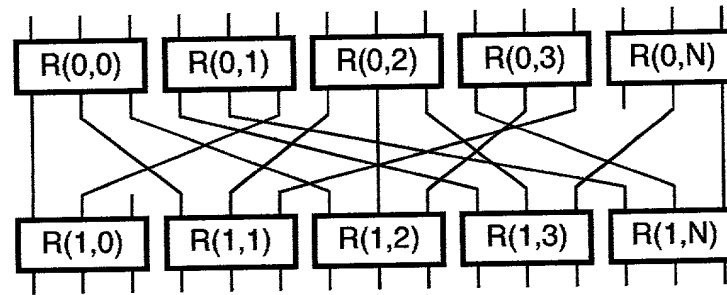


Fig. 6H

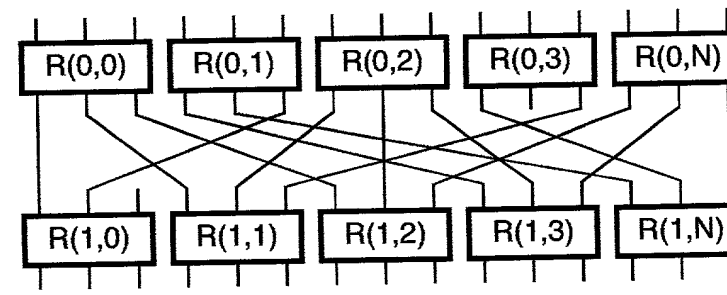


Fig. 6I

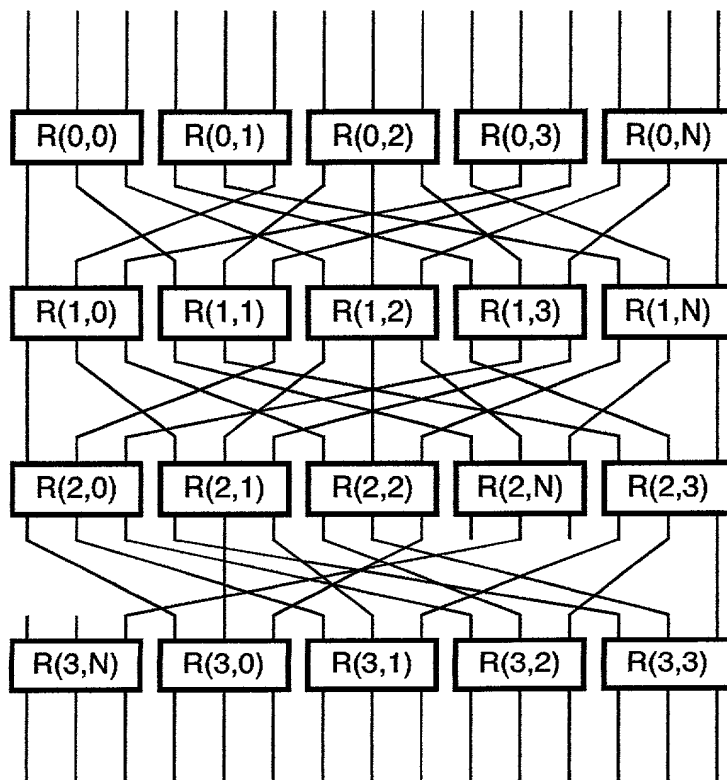


Fig. 6J

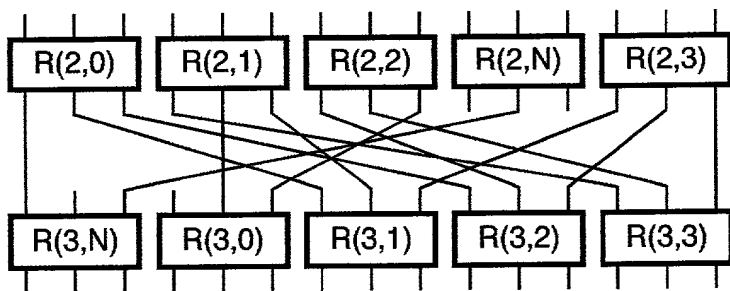


Fig. 7A

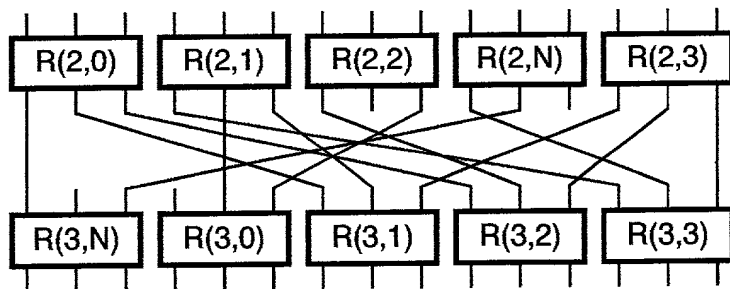


Fig. 7B

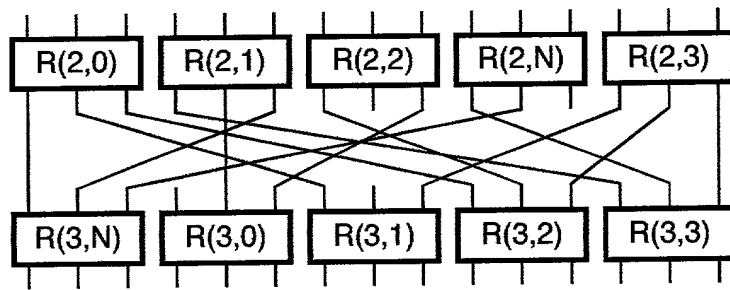


Fig. 7C

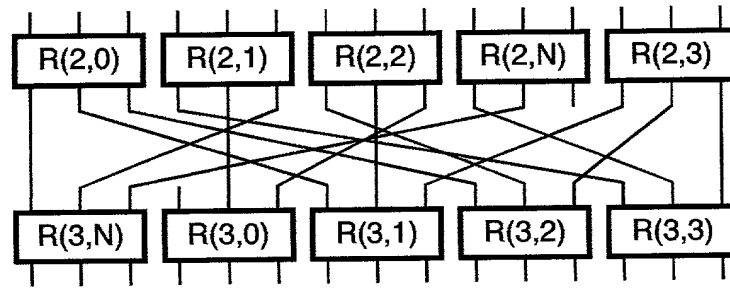


Fig. 7D

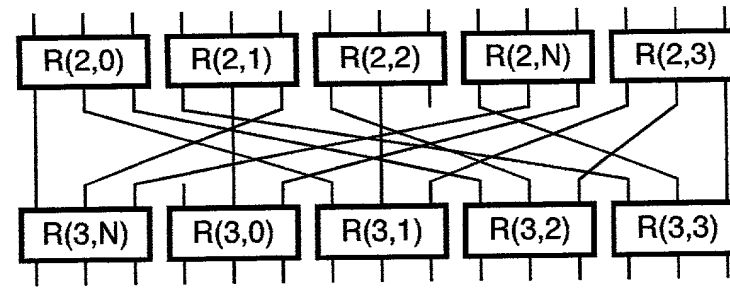


Fig. 7E

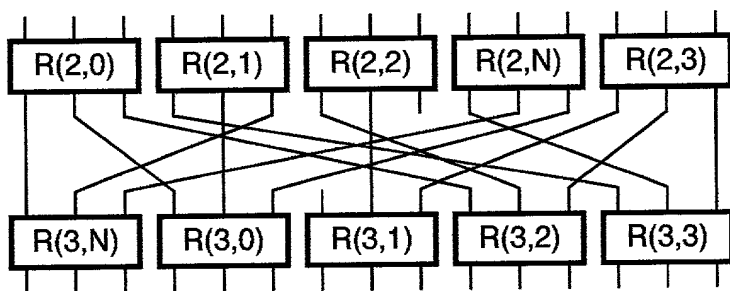


Fig. 7F

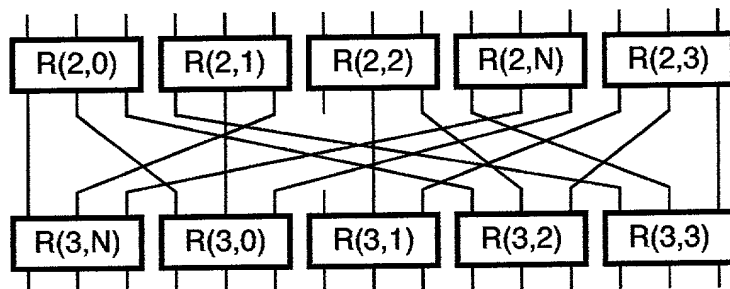


Fig. 7G

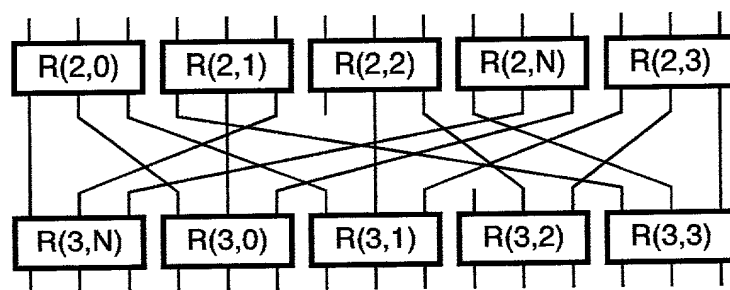


Fig. 7H

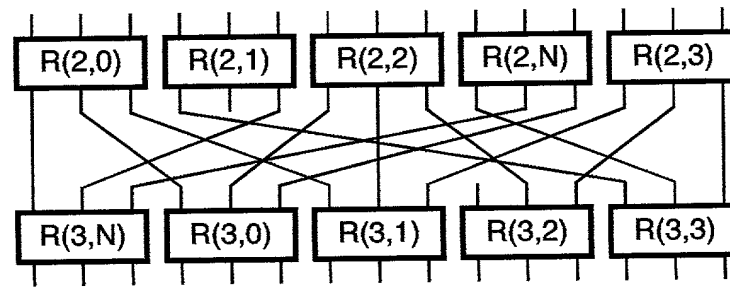


Fig. 7I

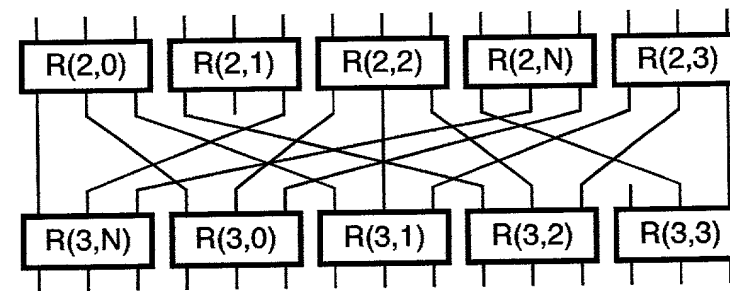


Fig. 7J

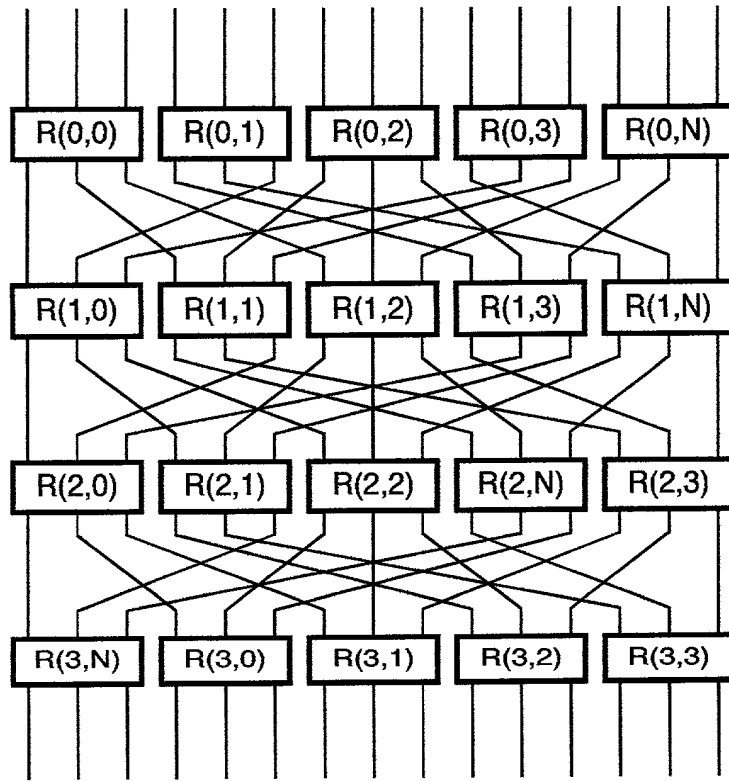


Fig. 7K

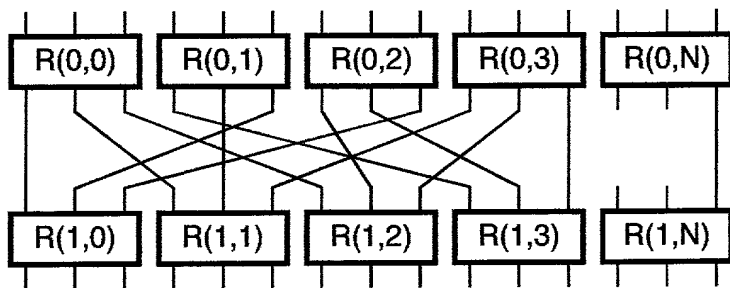


Fig. 8A

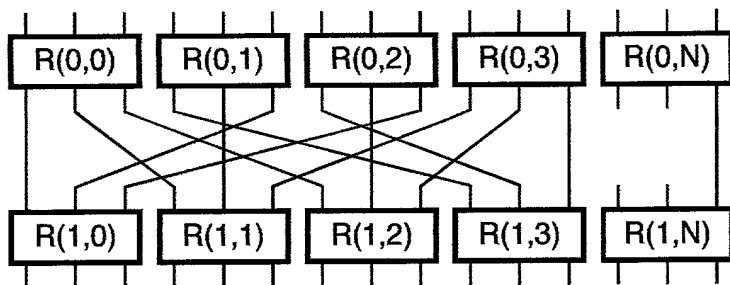


Fig. 8B

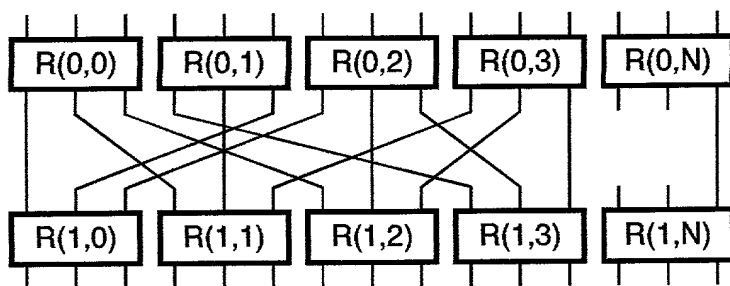


Fig. 8C

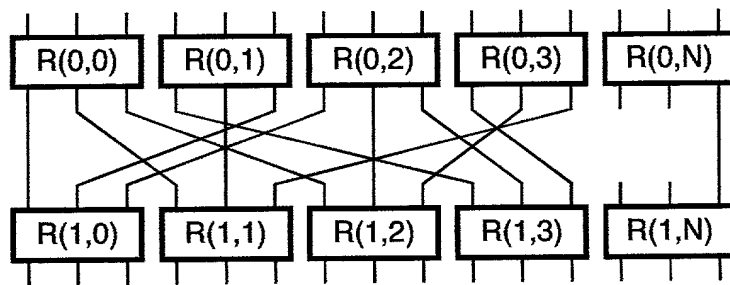


Fig. 8D

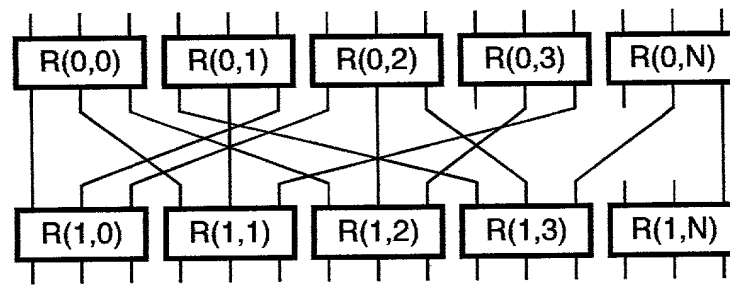


Fig. 8E

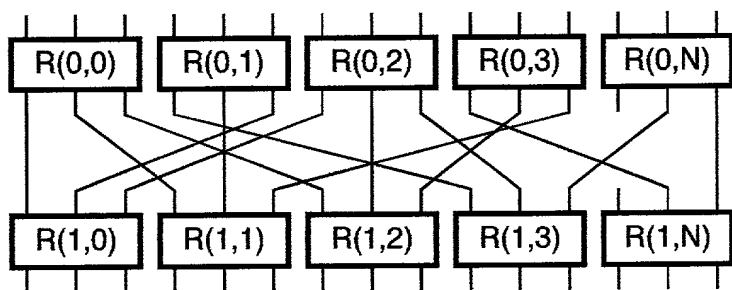


Fig. 8F

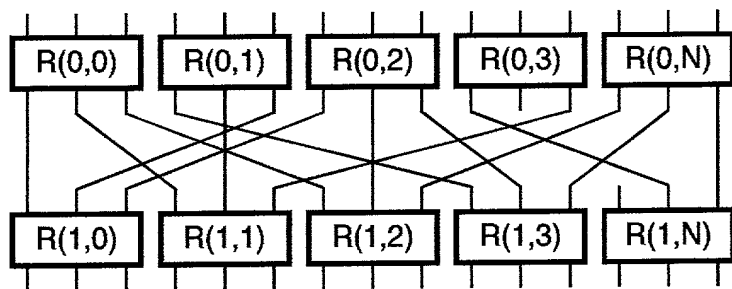


Fig. 8G

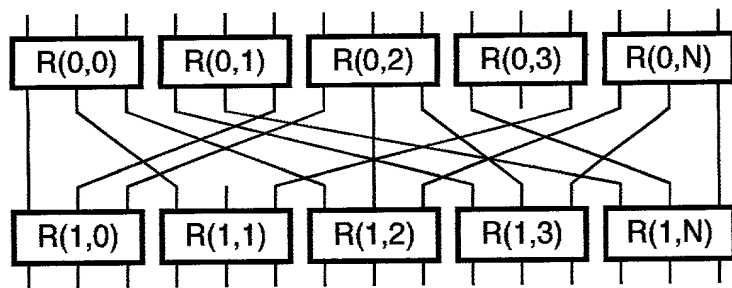


Fig. 8H

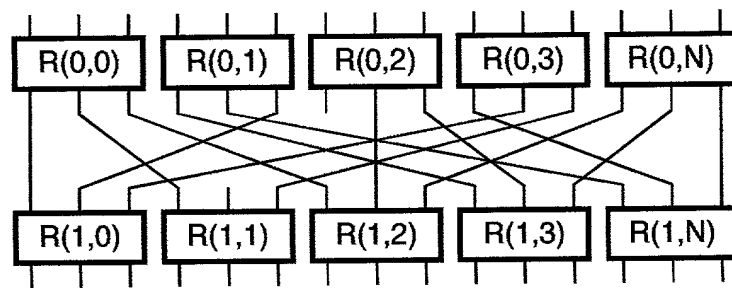


Fig. 8I

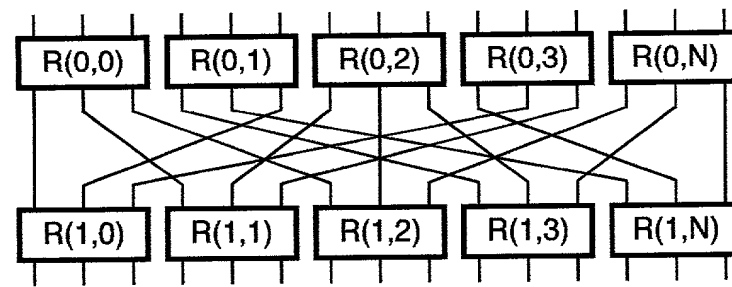


Fig. 8J


```

//
//Procedure Upgrade
//  Performs a width upgrade of a network, with num_rows total rows.
//  "want to relabel" may be "true" or "false" each time the statement is reached
//
Procedure Upgrade
for(current_row=0;current_row<num_rows;current_row++) {
  insert R(current_row,N) in position insertion_position(current_row);
  if(current_row>1) {
    iff(R(current_row-1,N) should be connected to R(current_row,N))
      Connect R(current_row-1,N) to R(current_row,N);
  }
}
do {
  start:
  if(want to relabel) {
    iff(any router, current_router, can be relabeled) {
      relabel_ports(current_router);
      goto start;
    }
  }
  select any port not connected to corresponding_port(port);
  target_port=corresponding_port(port);
  iff(target_port is already connected) {
    disconnected_port=port currently connected to target_port;
    Disconnect(target_port,disconnected_port);
  }
  Connect(port,target_port);
} while(there are misconnected ports);
connect_external_ports();
activate_external_ports();

```

Fig. 9A

```

//
//Procedure Upgrade
//  Performs a width upgrade of an RCCBG network with a fanout of fanout,
//  num_routers_per_row per row prior to width upgrade, and num_rows total rows. Also,
//  RELABEL_AVAILABLE flag if swapping of ports in a single router can be performed without
//  breaking connections.
//
Procedure Upgrade
for(current_row=0;current_row<num_rows;current_row++) {
    insert R(current_row, N) in position insertion_position(current_row);
    if(current_row>1) {
        if(R(current_row-1,N) should be connected to R(current_row,N))
            Connect R(current_row-1,N) to R(current_row,N);
    }
}

for(rindex=0;rindex<num_rows-1;rindex++) {
    current_row=row_select(rindex);
    if(RELABEL_AVAILABLE) {
        relabel_ports(current_row);
    }
    disconnected_port=None; // Holds the port previously disconnected by the last rewire step
    while((port=select_port(disconnected_port,current_row))!=None) {
        target_port=corresponding_port(port);
        if(target_port is already connected) {
            disconnected_port=port currently connected to target_port;
            Disconnect(target_port,disconnected_port);
        }
        Connect(port,target_port);
    }
}
connect_external_ports();
activate_external_ports();

```

Fig. 9B

```

//
//Simplification functions.
//
Function correct_port(port1,port2)
{
    if(port1 can be properly connect to port2)return(TRUE);
    else return(FALSE);
}
Function corresponding_port(port)
{
    if(port is a bottom port) {
        return top port of router in next row that should be properly connected to port port;
    } else {
        return bottom port of router in the previous row that should be properly connected to port port;
    }
}
Function Disconnect(port1,port2)
{
    Divert traffic away from port1 ;
    Divert traffic away from port2 ;
    Disconnect connection between port1 and port2 ;
}
Function Connect(port1,port2)
{
    Connect port1 and port2 ;
    Allow traffic to flow through port1 ;
    Allow traffic to flow through port2 ;
}

```

Fig. 9C

```

Function insertion_position(row) // Add a column version
{
    return(number_of_routers_per_row+1);
}

```

```

Function insertion_position(row) // For Fig.3B
{
    switch(row) {
        case 0:
            return 4;
        case 1:
            return 4;
        case 2:
            return 3;
        case 3:
            return 0;
    }
}

```

```

Function insertion_position(row) // For Fig.3C
{
    switch(row) {
        case 0:
            return 1;
        case 1:
            return 4;
        case 2:
            return 3;
        case 3:
            return 1;
    }
}

```

Fig. 10

```

Function row_select(row_index) {
    if(num_rows is even) {
        start_row=num_rows/2-1;
    } else {
        start_row=(num_rows-1)/2;
    }
    if(row_index is even) {
        return(start_row+row_index/2);
    } else {
        return(start_row-(row_index+1)/2);
    }
}

```

Fig. 11A

```

Function row_select(row_index) {
    return(row_index);
}

```

Fig. 11B

```

Function select_port(dport,current_row) // optimal dport is not used
{
    port_pool={port: bottom ports of routers in row, current_row and top port of routers in row,
                current_row+1 not connected to corresponding_port(port)};
    // For simplicity order right to left
    // First criterion
    for port in port_pool {
        if(disconnected(port) && disconnected(corresponding_port(port)))return(port);
    }
    // Second criterion: This basically says we prefer to target connections that break
    // connections only on fully populated routers
    for port in port_pool {
        if(disconnected(port) &&
            num_disconnections(router_of(port_connected_to(corresponding_port(port)=0))) {
            return(port);
        }
    }
    // Third criterion: Any port that is not connected
    for port in port_pool {
        if(disconnected(port)) return(port);
    }
    // Catch all for any ports left over: Not likely to be needed
    for port in port_pool {
        return(port);
    }
    return(None);
}

```

Fig. 12A

```

Function select_port(dport,row) // fill the hole
{
    if(dport !=None)return(dport);
    else {
        for all bottom ports, port, of routers in row current_row scanning from right to left {
            if(port is not connected to corresponding_port(port)) return(port);
        }
        return None; // No more ports to rewire
    }
}

```

Fig. 12B

```

Function select_port(dport,current_row) // round robin
{
    // This requires a FIFO of ports
    if(port_fifo empty) {
        port_fifo={port: bottom ports of routers in row, current_row and top port of routers in row,
                    current_row+1 which are disconnected};
    }
    if(port_pool empty) {
        port_pool={port: bottom ports of routers in row, current_row and top port of routers in row,
                    current_row+1 not connected to corresponding_port(port)};
        for port in port_pool {
            return(port);
        }
        // Catch all for any ports left over:Not likely to be needed
        port=any port not connected to proper port
        if(port exists) {
            return(port);
        }else {
            return(None);
        }
    }
    port=top of port_fifo ;
    remove top of port_fifo ;
    return(port);
}

```

Fig. 12C

```

Function relabel_ports(current_row)
{
    for(i=0;i<outers_per_row;i++) {
        for(bport1=0;bport1<fanout;bport1++) {
            for(bport2=0;bport2<fanout;bport2++) {
                //Test to see if the candidate port is connected to a router which one of the
                //other ports on the same router should be connected to.It doesn 't matter
                //at this point if it is the correct top port. That will be corrected in next loop.
                if(bottom port bport1 of R(current_row,i)is connected to any top port of
                router_of(corresponding_port(bottom port bport2 of R(current_row, i)) {
                    if(bport1!=bport2) {
                        exchange_ports(bport1 of R(current_row, i),bport2 of R(current_row, i));
                    }
                }
            }
        }
    }
    for(tport1=0;tport1<fanout;tport1++) {
        for(tport2=0;tport2<fanout;tport2++) {
            //Test to see if the candidate port is connected to a port which one of the
            //other ports on the same router should be connected to.
            if(top port tport1 of R(current_row+1, i)is connected to
            corresponding_port(top port tport2 of R(current_row+1, i)) {
                if(tport1!=tport2) {
                    exchange_ports(tport1 of R(current_row+1, i),tport2 of R(current_row+1, i));
                }
            }
        }
    }
}
//
//Auxiliary Procedures
//
Function router_of(port)
{
    return(the router which port belongs to);
}
//
//Here logical relabelling is assumed possible
//Other exchange schemes can be substituted
//
Function exchange_ports(port1,port2)
{
    permanently divert traffic originally intended for port1 to port2 ;
    permanently divert traffic originally intended for port2 to port1 ;
}

```

Fig. 13

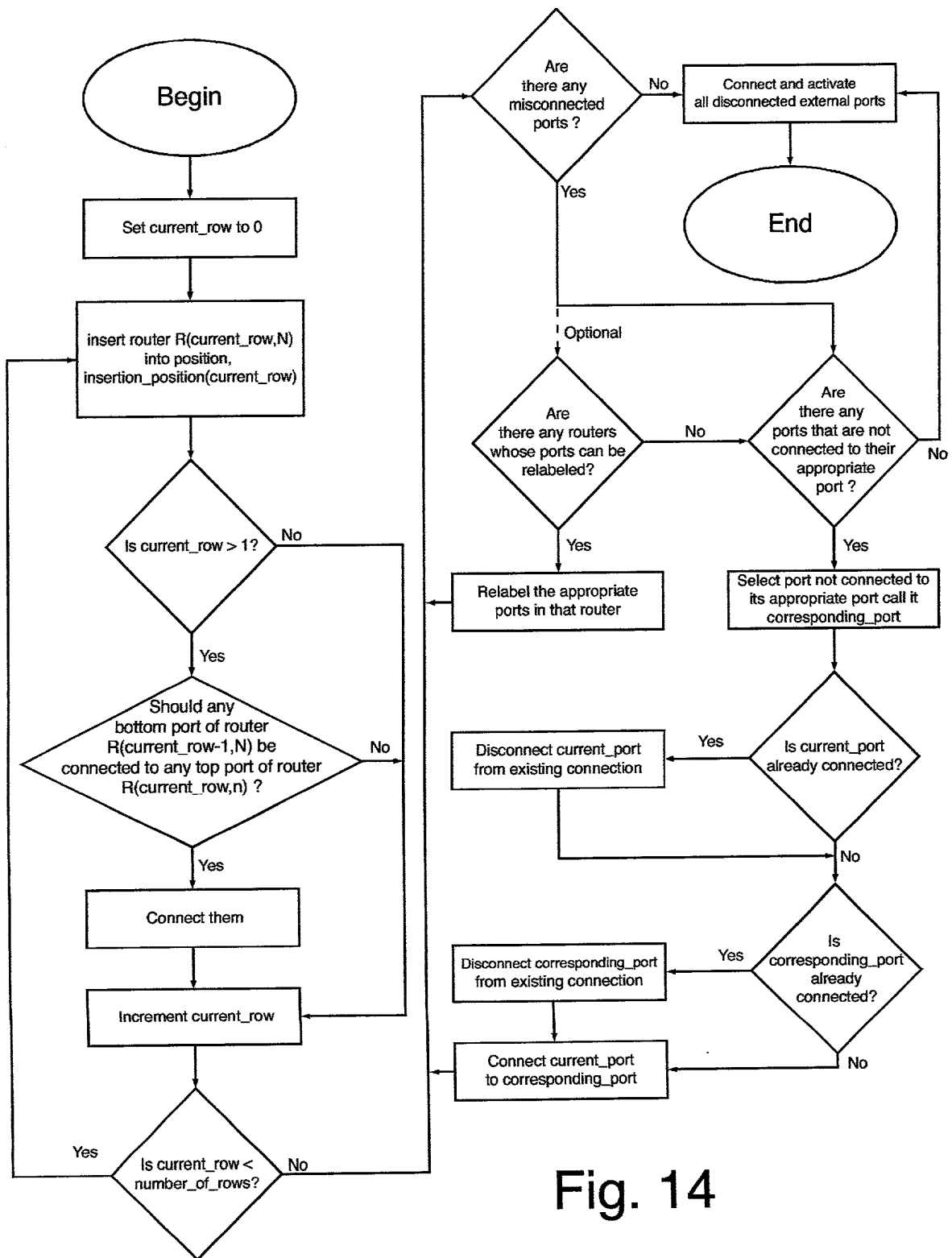


Fig. 14

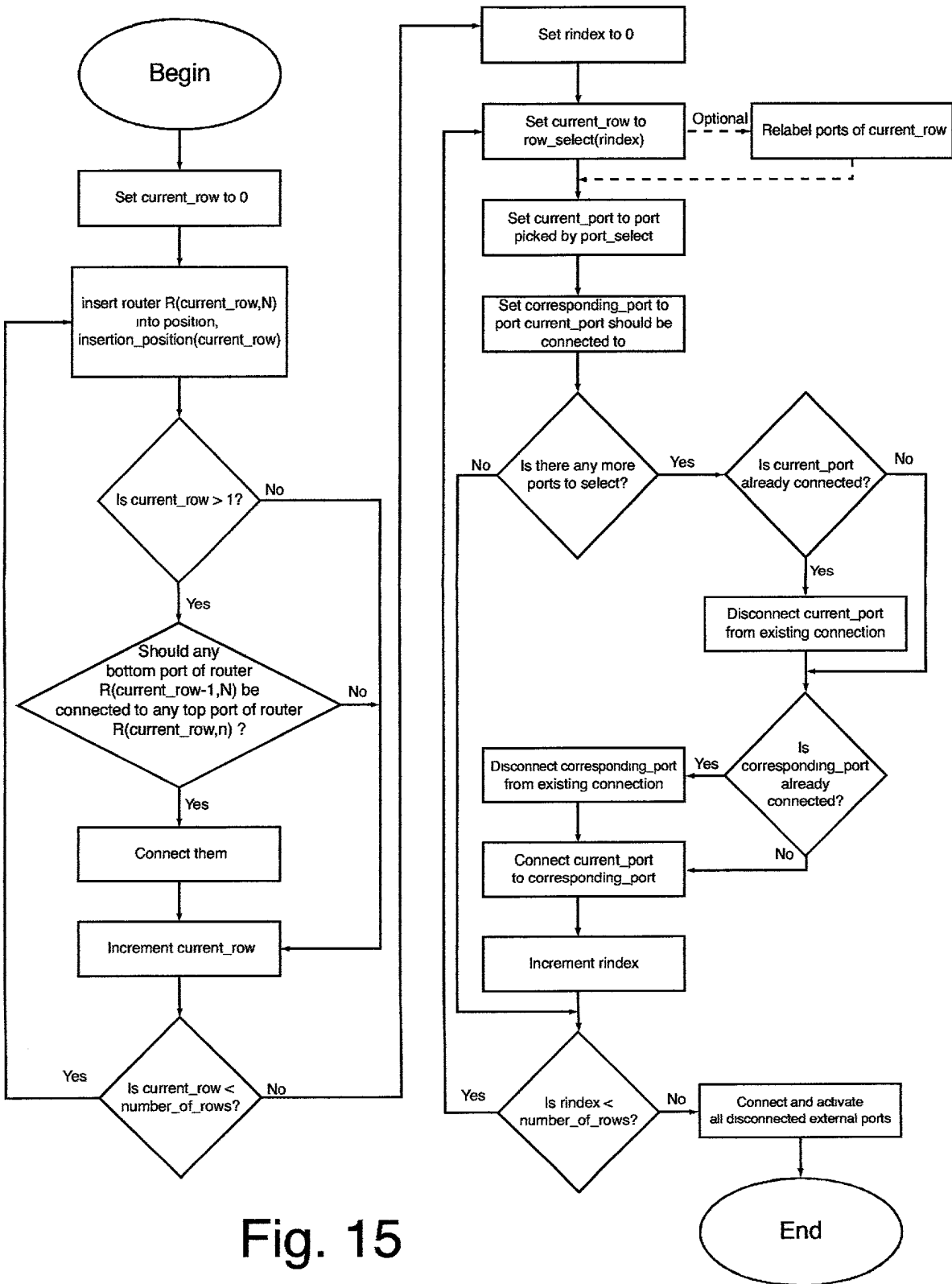


Fig. 15

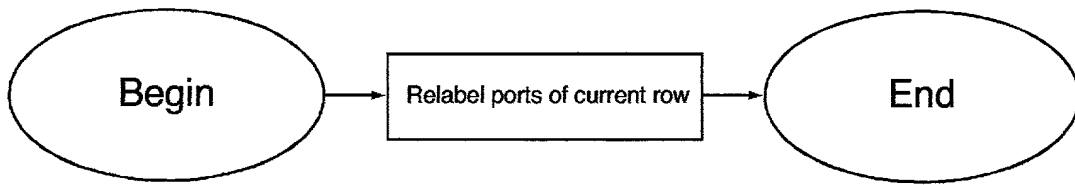


Fig. 16A

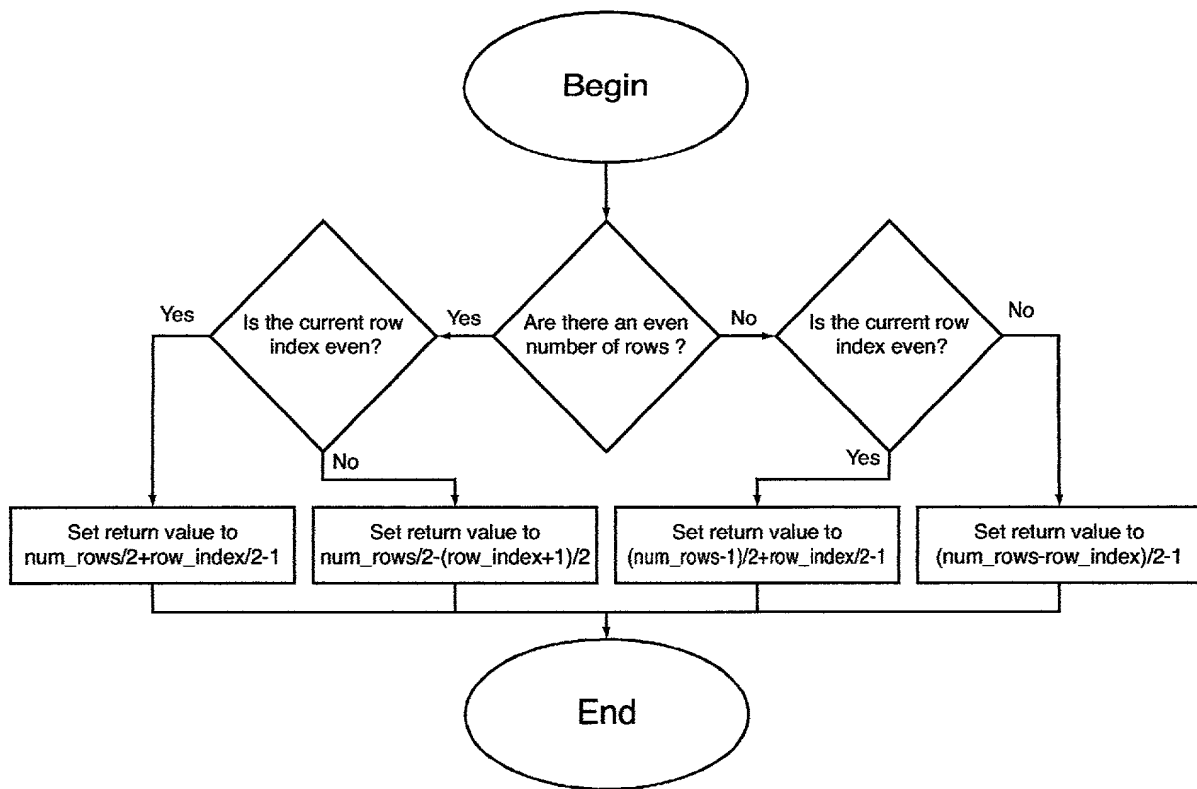


Fig. 16B

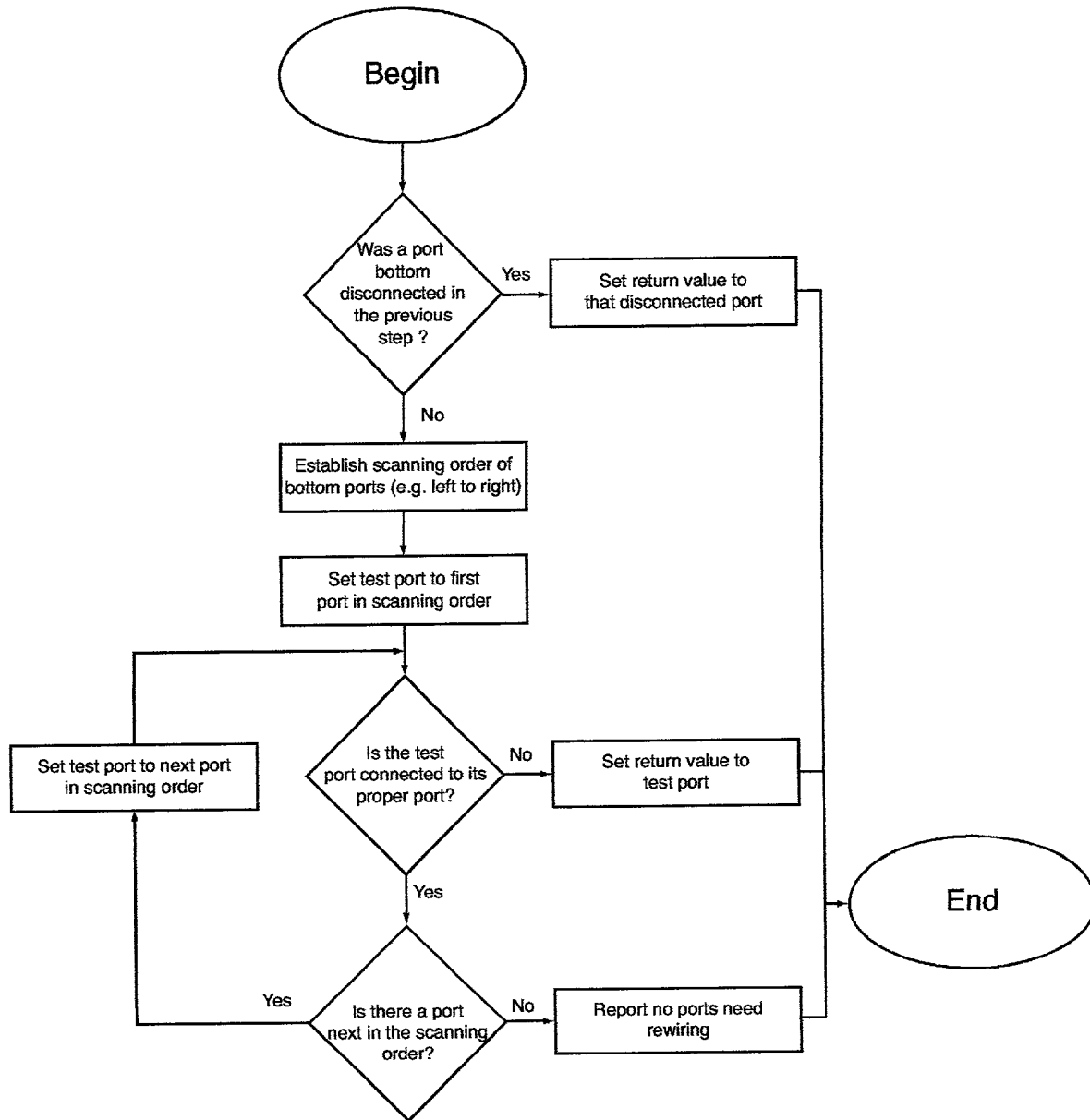


Fig. 17A

Fig. 17B

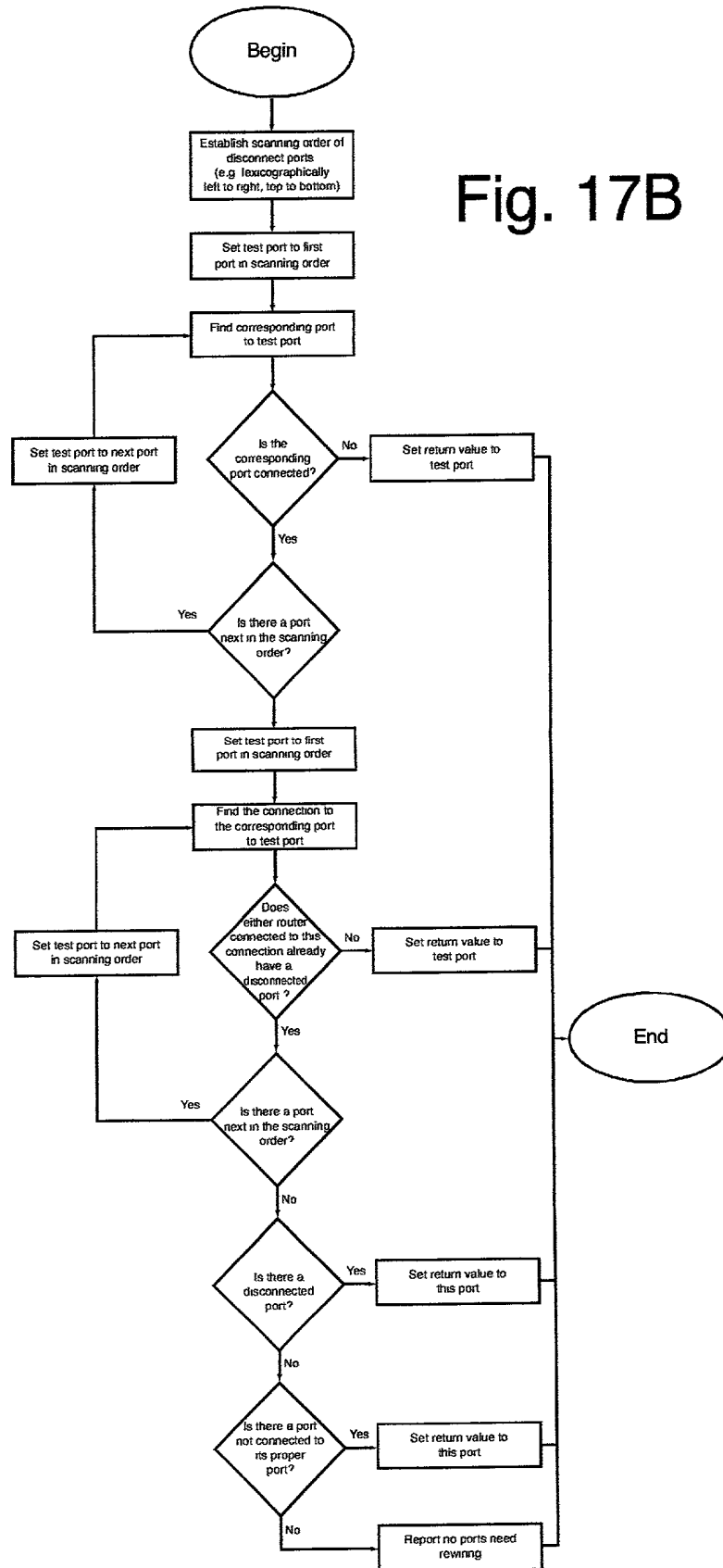
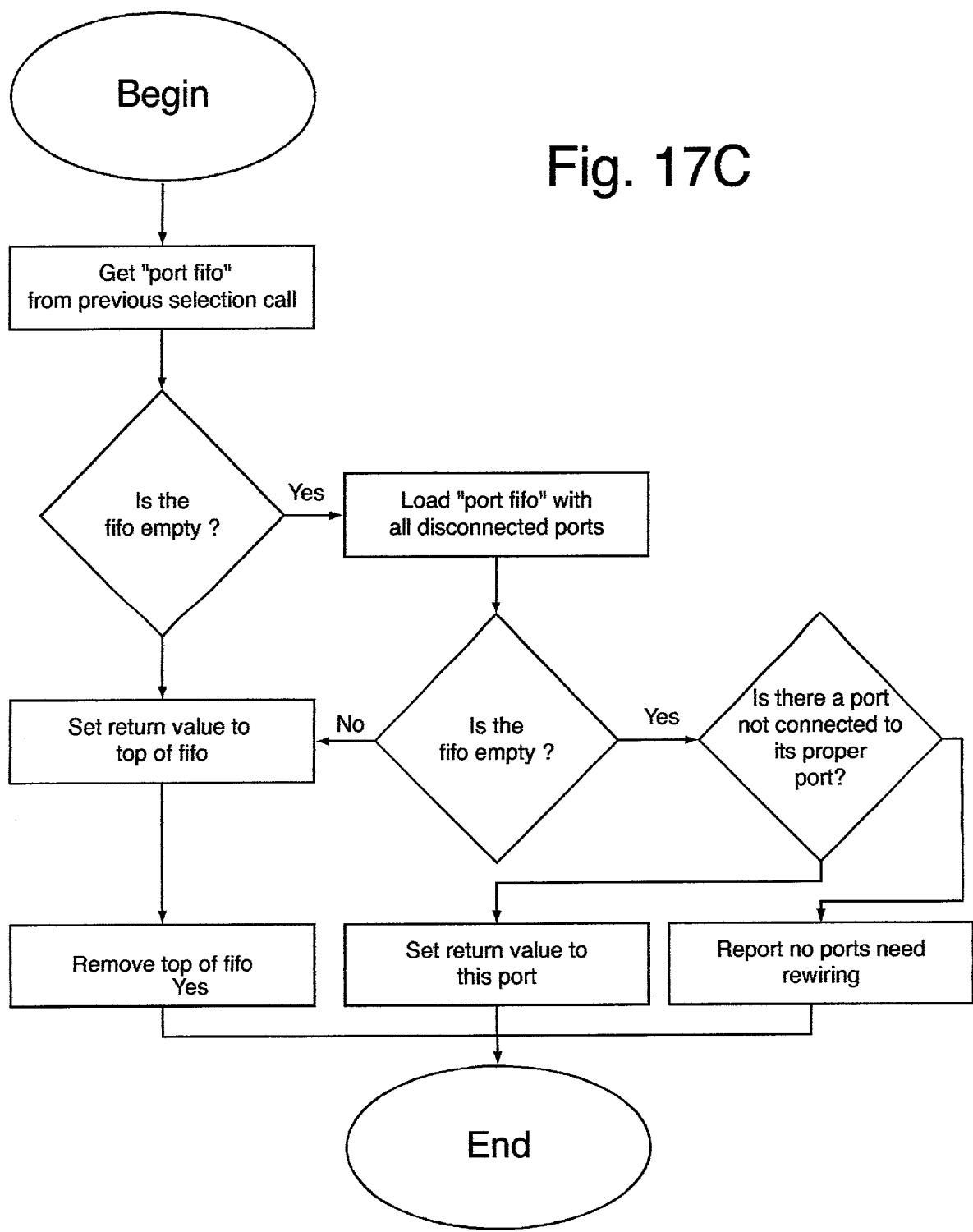


Fig. 17C



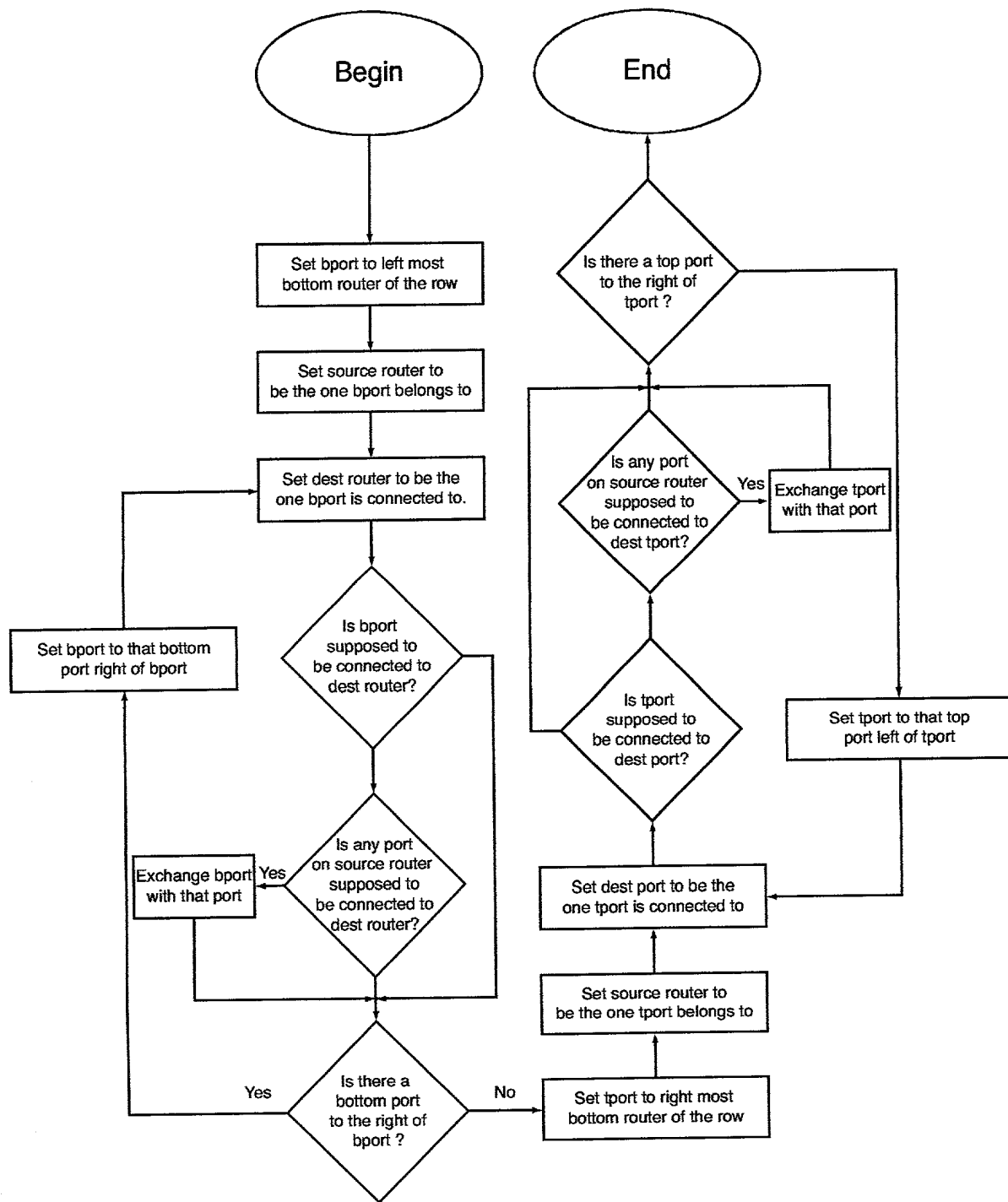


Fig. 18